


ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

# IMPLEMENTACIÓN DE UN TECLADO VIRTUAL MEDIANTE TÉCNICAS DE PROCESADO DIGITAL DE IMAGEN, EJECUTADO EN TIEMPO REAL SOBRE UNA BEAGLEBOARD

MEMORIA

Juan Luís Salaberri Coronado

Jeser Zalba Olcoz

Pamplona, 20 de Febrero de 2012

## *ÍNDICE*

# 1. INTRODUCCIÓN

## *1.1 Resumen del Proyecto*

Por el siguiente proyecto, se pretende crear un dispositivo que tenga las mismas funciones que un teclado al uso, pero a través de la visión por computador y sin elementos mecánicos como son las teclas.

Se quiere sustituir el concepto de “pulsar” una tecla, por el de “colocar” un dedo sobre la tecla deseada.

El sistema de manera autónoma recoge imágenes de un patrón previamente definido y obtiene datos acerca de la posición de un puntero, dedo u otro elemento que pueda apuntar hacia la “interrupción” (Entendiendo como interrupción, proceso por el cual la entrada de un periférico provoca una respuesta en el PC).
Se trata de conseguir a través de la visión por computador la relación “posición-respuesta”.

Este sistema se compone de una parte Hardware y otra Software.
Se pretende hacer un dispositivo independiente de un PC domestico. El objetivo de dichas pretensiones es la demostrar las capacidades tanto en el ámbito profesional como didáctico que puede tener el Hardware “BeagleBoard”, si lo combinamos con una distribución “Embebida” de Linux y las librerías de visión por computador “OpenCV”.

## *1.2 Objetivos.*

La visión por computadora deja de ser algo ficticio y hoy en día se consolida como algo esencial y necesario en muchos aspectos de nuestra vida.
Desde un lector de matrículas en un parking público hasta un algoritmo para la detección y seguimiento de la evolución de un cáncer de piel en el ámbito sanitario, pasando por control de masas de agua o el simple entretenimiento en los videojuegos.

La visión por computador debido al progreso en la calidad y rapidez de los procesadores ha sufrido una gran evolución.

El estudio de este campo ha originado una serie conocimientos básicos, algoritmos esenciales y aplicaciones comunes.

Durante la carrera, la Visión por computador fue estudiada y analizada con ayuda de un software de Cálculo Matemático “Matlab”. Una de las grandes ventajas de este software era la facilidad y potencia de cálculo que tenia de cara a operar con vectores.
Y teniendo en cuenta la información de las imágenes como grandes vectores hacia de ”Matlab” una gran herramienta didáctica para el análisis del campo de la visión por computador.

A la hora de plantear este proyecto se tuvo en cuenta dicho trabajo y se propuso el análisis de este campo a través de otras herramientas.

Se vio en las bibliotecas “OpenCV” y en la placa “BeagleBoard” una buena oportunidad para el trabajo de análisis e investigación.
“OpenCV” es una librería libre de visión artificial, desarrollada por “Intel”. Es publicada bajo licencia BSD, lo que le permite ser usada libremente para usos didácticos e investigación. Pretendiendo ser un entorno de desarrollo fácil de usar y eficiente.

Por otro lado la placa “BeagleBoard” se distribuye como computadora de baja potencia y bajo coste con fines didácticos.

Si juntamos las características de ambos elementos podemos considerar un dispositivo de bajo coste y alto rendimiento ideal para su uso didáctico y de investigación.
Se podría plantear como una herramienta eficiente para la enseñanza de este campo

Para el desarrollo final se planteó una aplicación que explotara lo máximo posible las funcionalidades de este conjunto.
Finalmente se dio con un concepto de teclado que poco a poco puede llegar a ser una realidad en los dispositivos móviles de hoy en día.

Algunos fabricantes de dispositivos móviles ya incluyen este sistema en los prototipos de sus futuros productos (Mozilla concept mobile “Seabird”, Iphone 5 concept…).

## *1.3 Descripción de la planificación.*

Se plantea por tanto la posibilidad de agrupar en un dispositivo las capacidades técnicas de la placa “Beagleboard” con las posibilidades funcionales de las librerías “OpenCV”.

Se plantean desde un principio las diferentes posibilidades que existen; Se propone usar tanto un sistema operativo comercial como es Windows y un Sistema operativo de código abierto como es Linux.

Se pretende demostrar la versatilidad que ofrece la placa y la posibilidad de poder tener accesible más de un sistema operativo con el simple hecho del cambio de Tarjeta SD.

Por otro lado se plantea también la posibilidad de variar los resultados entre diferentes cámaras-web. Probar así diferentes resultados y los puntos flacos que el algoritmo pueda tener en ese aspecto.

Finalmente a la vista de las dificultades en la programación con C++ se lleva a cabo el diseño del programa en otro Software, en este caso, Matlab.
A través de este paso intermedio se pretende demostrar el correcto funcionamiento de los algoritmos que compondrán el programa final así como el proceso entero, desde que se calibra el teclado hasta que se obtiene un resultado en forma de coordenadas en el plano “real”.

# 2 DISEÑO Y ARQUITECTURA DE HARDWARE.

Se pasa a describir los diferentes elementos físicos (Hardware) que componen el dispositivo.

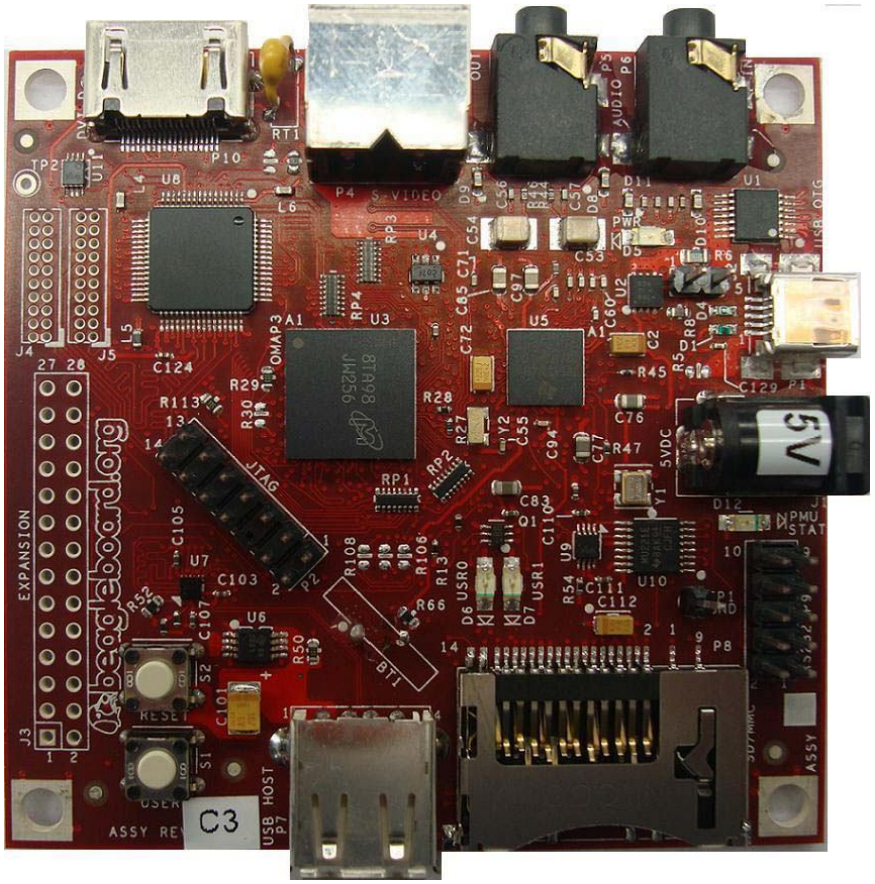
## *2.1 Beagleboard*

Beagleboard es el nombre que adquiere la placa base de alto rendimiento y baja potencia comercializada por Texas Instruments en asociación con "Digi-Key". Fue diseñada con software "libre" y desarrollada por un reducido equipo de Ingenieros de Texas Instruments con fines didácticos. Pensada para uso en ámbitos universitarios para la enseñanza de Hardware y Software libre. Su principal idea se basa en la demostración de las capacidades del sistema OMAP (Open Multimedia Application Platform).

Este a su vez es una categoría contenida en el "System on Chips" (SoCs), es decir, circuitos integrados que contienen todos los componentes de una computadora o sistema electrónico. Especialmente diseñados para sistemas portátiles y tecnología móvil.

Su arquitectura soporta tanto sistemas Windows como Linux.

Durante todo el desarrollo de este proyecto se ha usado la revisión C3 de la placa.
Esta variante aparecida en 2009 incorpora ligeras mejoras respecto de su antecesora, la C2.



Las características principales de la placa Beagleboard, revisión C3, son:

- Dimensiones: 7.62 cm x 7.62 cm x 1.6 cm
- Peso: <40 gr
- Memoria: 256 MB
- Frecuencia: 600 MHz.
- Sistema de alimentación:
    - Fuente de 5V en continua.
    - USB desde otro PC.
    - Limitado a 500 mA
- Puerto USB. (Necesidad de HUB).
- Puertos Minijack de entrada y salida. Salida de color de 24b.
    - NOTA: No soporta Full HDMI, simplemente se eligió este interfaz por su reducido tamaño. Es necesario un adaptador HDMI/DVI-D.
- Puerto HDMI para salida de video a monitor.
- Slot para tarjeta SD. Será el utilizado para dar alojo al sistema operativo y Software. Aunque también es capaz de dar soporte a cámaras de video, tarjetas bluetooth, módulos GPS y tarjetas WIFI.
- Dos botones accionables por el usuario:
    - RESET: Causa un reseteo total de la energía.
    - USER/ BOOT BUTTON: Esta destinado a dos funciones, principalmente:
        - Forzar el cambio de "boot-eo" en la secuencia de inicialización.
        - Como botón de aplicación programable por software.

  Al pulsar el botón "user" en la secuencia de inicialización, el sistema cambia la secuencia de "boot-eo" al siguiente orden; USB, UART (Puerto Serie), MMC1, NAND.
- Dos Leds indicadores:
    - Arranque.
    - Lectura.
    - Bloqueo.

A lo largo del desarrollo de la placa, el fabricante renueva la tecnología de la misma y cada cierto tiempo aparecen actualizaciones del hardware.

Durante el transcurso de este proyecto aparecieron en el mercado dos versiones diferentes de la Beagleboard.

Diferencias respecto a la **versión anterior**: (C2)

- Muescas para tornillos conectadas a tierra.
- Interface S-VIDEO mejorada.
- Slot para baterías de Litio incorporada.

Diferencias respecto a la **versión posterior** (C4): (Diciembre de 2009)

- Mejora en el procesador OMAP
- Actualización del software de Boot-eo (Uboot).
- Mejora de aislamiento de ruido eléctrico en la parte de alimentación.

De cara al presente proyecto el desarrollo que se realizo en la placa fue sobre todo el trabajo sobre el sistema operativo que iría alojado en las tarjetas SD.
Esto supone ligeras modificaciones en la BIOS del sistema. Era necesario configurar la manera por la cual la placa generaba la señal de video.

Dejando a un lado el trabajo sobre el sistema operativo, los trabajos se encaminaron en modificar las líneas de "Boot-eo" con el fin de que la señal de video fuera la correcta. En definitiva, determinar en la placa que la señal de salida de video fuera a través del puerto HDMI y con una resolución 800x600.

Además es necesario determinar que el siguiente paso en la secuencia de inicialización ha de ser la lectura del sistema operativo en la tarjeta SD. Esa cuestión será analizada más a fondo en el apartado dedicado al Sistema Operativo.

## 2.2 Cámara.

Desde un principio se vio la necesidad de trabajar con periféricos que no fueran demasiado voluminosos para mantener la naturaleza "móvil" del dispositivo, además con la idea de no incrementar gastos, se decidió trabajar con periféricos que pudiéramos reutilizar en casa de otros dispositivos.

Se utilizaron dos cámaras web distintas:
- Classic Silver (Hercules).
- EyeToy para Play Station 2 de Namtai.

La primera es una cámara web al uso como cualquier otra que podemos adquirir en las distintas tiendas especializadas.

Tiene una resolución de 1,3 MP y 30 fps, se alimenta por USB e incorpora Micrófono.
Una de las características que la hacían útil, era que disponía de una "Pinza" para poderla colocar en diferentes lugares.

La segunda cámara es más compacta en cuanto a tamaño se refiere. Se trata de la cámara que comercializó Sony para su Play Station 2, para poder jugar a los juegos en los que se interactuaba a través de los movimientos del cuerpo.

Se trata de una cámara por USB que pesa 173 gramos y tiene una resolución de 320 x 240.
Trabaja con el sensor OV7648 de "Omni Vision".
Tiene un tamaño de lente de ¼" y una sensibilidad para el color de 2,20 V/Lux-sec.

A lo largo del desarrollo del dispositivo, se han realizado pruebas con las distintas cámaras y se han analizado resultados.
Al final se concluye lo siguiente:

Debido al trabajo que supone programar y realizar pruebas de funcionamiento además del coste computacional que supone trabajar con video y compilar código c++, se decidió trabajar con un PC de sobremesa en el que hacer las pruebas de código y funcionamiento correspondientes. Así cuando un código era estable y ya estaba compilado, se pasaba a la Beagleboard y se realizaban las últimas pruebas allí.

Así es como se descubrió que dada naturaleza embebida del Kernel Linux que se destina a la Beagleboard, había ciertos problemas de compatibilidad entre las cámaras y la propia "Beagle". Se observo que la Beagleboard no era capaz de tratar la información que provenía de la cámara "Hercules", correctamente. Finalmente y tras una reinstalación del Kernel y de las librerías OpenCV se comprobó que, efectivamente, la cámara "Hercules", no funcionaba correctamente en la Beagle por lo que se decidió trabajar únicamente con la cámara "EyeToy".

Esto conlleva otras lecturas. El hecho de hacer funcionar el programa con las distintas cámaras, ha llevado a resultados diferentes. El código final se diseño con el fin de ser lo más robusto posible a los cambios circustanciales. Pero la diferencia de un sensor a otro y de una lente a otra hacía imposible un mismo código para ambas cámaras.

Hay que tener en cuenta por un lado, que las cámaras tienen distinta sensibilidad a la luz, y cada una tiene una manera de auto-ajuste ante las variaciones de luz. Es por ello que se comportan de diferente manera ante una ventana abierta o una sombra que pueda interponerse entre la fuente de luz y el patrón.

Por otro lado tenemos que ambas cámaras tienen una lente de diferente distancia focal, por lo que la imagen que capta una y otra es diferente. En una tendremos un campo de visión más reducido por lo que necesitaremos más espacio entre lente y patrón para que pueda ser captado en su totalidad.

Esto nos lleva a que el algoritmo tenga ligeras modificaciones cuando se trataba con una cámara u otra. Parámetros como los umbrales de binarizacion, o criterios a la hora de discretizar elementos conexos, así como la frecuencia de refresco de las imágenes, eran necesarios revisarlos antes de considerar resultados satisfactorios.

Esto llevado a un supuesto producto final, podría dar pie al estudio de un hardware dedicado, las posibilidades de sensores de imagen son infinitas por lo que sería difícil precisar un único software para todas las cámaras comercializadas.

## *2.3 Otros periféricos.*

Además de la Beagleboard y la cámara, este dispositivo requiere de otros periféricos para poder funcionar correctamente.
Este dispositivo no trata de ser un producto final. Si no más bien un concepto o prototipo de las capacidades que puede llegar a tener un Hardware como la Beagleboard.

Es por ello que no se le ha dado toda la autonomía a la aplicación y es por ello que se necesario después de todo la ayuda de un teclado o ratón para poder ejecutar el programa y poder darle alguna orden concreta. Los teclados y ratones usados han sido los mismos que se usan para todo tipo de PC.

Finalmente destacar también la necesidad de una pantalla, en la que poder revisar el transcurso de los resultados.
Esta debe ser de entrada DVI-D para que pueda ser usada junto con la BeagleBoard.

# 3 DISEÑO Y ARQUITECTURA DE SOFTWARE

Como objetivo a lo largo del proyecto estaba la tarea de experimentar las capacidades del hardware y todas las variantes que podría tomar la combinación de OpenCV junto con diferentes sistemas Operativos.

Es por eso que se realizaron pruebas bajo distintas plataformas, no siendo fructíferas todas ellas.

Además el diseño previo y pruebas de la aplicación final de Teclado Virtual, se realizaron sobre Matlab, debido a que era un lenguaje más conocido y más rápido a la hora obtener los resultados.

## *3.1 Pruebas iniciales.*

En un principio se instalaron las librerías en un PC de sobremesa, con el fin de empezar a trabajar y probar la programación con OpenCV. Pero la inexperiencia con C++ y OpenCV hizo que el trabajo fuera más lento de lo deseado.

Finalmente se optó por implementar y probar que los algoritmos que se habían planteado al empezar el diseño de la aplicación eran fructíferos realizando la programación completa sobre Matlab, y una vez demostrado, pasaríamos a traducir el código en OpenCV.

### 3.1.1 Matlab

Matlab es un software matemático que ofrece un entorno de desarrollo integrado. (IDE), y tiene un lenguaje de programación propio. Puede correr sobre plataforma Windows, Unix y MacOS.

Su gran prestación es el trabajo eficiente con matrices y vectores, así como la representación de gráficos, datos y funciones.

Sin embargo el elemento que hace clave a este software para la programación de nuestra aplicación es el Toolbox de tratamiento de Imágenes que podemos añadir.
Es un software muy empleado en entornos de investigación y desarrollo, como son Universidades y centros de Investigación.
Para este proyecto fue usada una distribución instalada sobre un PC de sobremesa proporcionado por la Universidad, en Su versión R2010a De Marzo de 2010.

Siguiendo este método sabríamos que los procesos intermedios han de entregar resultados satisfactorios y evitar así que cualquier fallo de programación pudiera dar resultados “falsos”.

Prácticamente el proceso que sigue la aplicación para obtener los resultados es el mismo que en el caso de OpenCV. Únicamente se le ha dotado a la aplicación final en C++ de más autonomía y menor manipulación por parte de un usuario externo. Así, el teclado virtual en OpenCV es capaz de detectar cuando se ha introducido un dedo y pasar el, automáticamente, a realizar el cálculo de la posición.

Hay ciertos matices y diferencias que se detallaran en el desarrollo de la aplicación. Pero se puede afirmar que OpenCV y Matlab arrojan resultados similares, si bien no se han hecho cálculos de velocidad al tratarse el objetivo de conseguir una aplicación embebida, y como hemos mencionado anteriormente a la aplicación en OpenCV se le ha dotado de mayor autonomía que en el caso de Matlab.

## *3.2 Implementación Beagle Board*

### 3.2.1 Sistema Operativo

Como se ha mencionado la placa Beagleboard soporta diferentes Sistemas Operativos. Dentro de la comunidad de desarrolladores han demostrado que funciona correctamente bajo Android, Angstrom, Fedora, Ubuntu, Gentoo y WindowsCE.

En nuestro caso realizamos las pruebas con Linux (Ubuntu) y WindowsCE. En el primer caso la implementación fue completa, mientras que para el caso de Windows, no se pudo instalar OpenCV y por consiguiente no se pudo desarrollar la aplicación sobre esa plataforma.

- Windows CE

Windows CE o “Windows Embbeded Compact” es un sistema independiente desarrollado por Microsoft para plataformas embebidas o dedicadas.

La naturaleza comercial del producto hace que las posibilidades de manipulación sean reducidas y la mayoría de posibilidades de desarrollo requieren un coste económico.
Se ha buscado información y soporte en diferentes paginas. Toda esa información acaba en empresas de desarrollo de software que piden una cuantía económica por una distribución de Windows CE que funcione sobre Beagleboard. Este fue el caso, por ejemplo de MPC DATA. La cual, sin embargo, permitía descargar una distribución en modo demo con las características básicas para el correcto funcionamiento sobre la placa, pero que no permitía por ejemplo la instalación de OpenCV.

La creación de una versión embebida de Windows CE, necesita de la plataforma Visual Studio y es necesario un nivel avanzado de programación en Visual así como ciertos conocimientos de desarrollo software.

Por todo esto se ha llevado a hacer pruebas con productos ya finalizados en versión Demo que eran proporcionados por ciertas compañías.

Sin embargo esto suponía la no posibilidad de modificar dichas versiones del S.O a nuestras necesidades, por lo que la incorporación de OpenCV a nuestro sistema Windows no fue posible.

El proceso seguido para la instalación y funcionamiento de Windows CE sobre la Beagleboard fue el siguiente:

- Descargada la documentación necesaria.
- Se formatea la tarjeta SD de 2 Gb con la ayuda del software de formateo de HP. (Dentro de la carpeta de "Recursos", apartado "bin").
- Es importante copiar los siguientes archivos en el siguiente orden: (Todos los archivos se encuentran en la carpeta "recursos" en el apartado "bin" en la parte correspondiente a la tarjeta SD.)
    - MLO
    - Ebootsd.nb0
    - Nk.bin
- Una vez hecho esto, podremos reiniciar la placa y veremos que WindowsCE arranca sin problemas. Como ya hemos comentado con recursos reducidos.


Se realizo con éxito la incorporación de Windows CE a la Beagleboard en una tarjeta SD distinta a la que incorporaba Ubuntu, pero sin embargo tras varias semanas de trabajo, no se pudo añadir librerías OpenCV a dicho sistema.

Es por ello que la posibilidad de usar librerías OpenCV sobre Windows CE y Beagleboard se ha dado por descartada, lo cual sin embargo no significa que no haya posibilidad de hacerlo ya que existe diversa documentación al respecto y las comunidades de desarrolladores continúan con dicha labor.

- Ubuntu

Dentro de las distribuciones disponibles para poder trabajar junto con al BeagleBoard se escogió Ubuntu por ser una distribución sencilla de configurar y necesita de pocos recursos del hardware si se le instala una versión de escritorio de características mínimas como es el caso del XFCE.

Dentro de las documentación encontrada en Internet acerca de Linux en Beagleboard. Ubuntu sea quizás la que mas soporte tenga. Además en previsión de la utilización de OpenCV. Dentro de la propia página de descarga de OpenCV se afirma que la instalación con menos problemas conocidos y más documentación al respecto es la asociada a Ubuntu.

Ubuntu, que toma su nombre de una ideología Sudafricana (Ubuntu: igualdad/Lealtad hacia otros) es una distribución Linux mantenida por una empresa de dueños Sudafricanos, Canonical. Está basada en Debian y hoy en día es quizás la distribución Linux mas empleada por el usuario medio.

Ubuntu está orientado a la facilidad de uso y soporte y a mejorar la experiencia de usuario.
Cada 6 meses está estipulado que aparece una nueva versión. La última versión de Ubuntu instalada en la Beagleboard data de Octubre y corresponde a la versión 11.10, conocida como Oneiric.
Como ya se ha comentado, una vez instalado Linux, en nuestro caso se opto por la instalación de un Escritorio de pocos recursos, el XFCE.

Existen varias opciones para conseguir una imagen de Linux funcionando en la Beagleboard. Podemos usar imágenes totalmente operativas ya compiladas colgadas en internet de manera gratuita las cuales son posibles modificar. Se puede también compilar una propia versión de Ubuntu a través de una serie de herramientas que es necesario instalar en un PC secundario. Y Finalmente existe la posibilidad de descargarse una serie de Scripts que automatizan las tareas de formateo y descarga de imágenes de Ubuntu desde internet. Estos scripts se encuentran en la carpeta de recursos en el apartado “Ubuntu”.

Lo primero de todo es tener una tarjeta SD de 2 Gb (recomendado), y formatearla de manera que nos queden 2 particiones. El formateo se realizo con el software de Linux (fdisk) para tal función. La primera de 50 MB será la que contenga los archivos de arranque y la que se comunique con el hardware a la hora de arrancar el sistema operativo. Por otro lado tendremos la partición, con el espacio restante, en el que irán alojados los archivos del sistema operativo propiamente dicho.

A lo largo del desarrollo del proyecto se han probado las distintas opciones y se concluye que los diferentes procesos llevan a resultados similares. Sin embargo unos procesos son más complejos de llevar a cabo que otros. Además el tiempo necesario para completar una u otra instalación es diferente.

La última versión instalada en la SD que se ha llevado hasta el día de hoy ha sido instalada de la siguiente manera:

- Debemos tener formateada, tal y como se ha mencionado, la tarjeta SD en dos particiones distintas.
- A continuación deberemos de copiar una serie de archivos en el orden indicado sobre las dos particiones.

    - **MLO:** Es un pequeño gestor de arranque de primer orden. Debido a que la RAM estática de los sistemas embebidos es muy pequeña, MLO es compilado con lo esencial; inicializa memoria y carga la cantidad necesaria y mínima para acceder cargar el gestor de segunda etapa, u-boot.
    - **U-boot:** El u-boot es un gestor de arranque universal para placas embebidas ARM que puede ser instalado en una ROM de arranque y se utiliza para inicializar hardware y ejecutar sistemas operativos.
    - **U-Image:** Es la imagen del Kernel de Linux compilada.

- Seguidamente nos descargamos una imagen ya compilada de la propia página de soporte de Beagleboard. (Las imágenes se encuentran en el CD de recursos dentro del apartado de “Ubuntu”).

Una vez están estos tres archivos dentro de la BeagleBoard deberemos descomprimir la carpeta que contiene los archivos del sistema operativo dentro de la propia tarjeta y en la segunda partición. Esta carpeta habrá sido previamente copiada en dicha partición.

Finalizado este proceso podremos borrar la carpeta comprimida.
En este punto deberíamos ser capaces de arrancar Linux en la Beagleboard.

Una vez arrancado Linux, nos damos cuenta de que no tenemos interfaz grafica y que únicamente podemos visualizar el Shell de Linux.
Es necesario configurar la red y conectar la placa a internet para poder instalar el escritorio y entorno grafico.

Este proceso de instalación requiere de bastante tiempo ya que la cantidad de operaciones que la placa es capaz de hacer por segundo esta algo limitada. El procesador no es tan potente como en un ordenador al uso.

Una vez completada la instalación, disponemos ya de Ubuntu con su entorno grafico en la Beagleboard. Ahora ya si, podemos tratar al sistema como un pequeño ordenador, en el que podremos instalar nuestras librerías OpenCV y poder así compilar aplicaciones de visión artificial.

Es necesario comentar que a pesar de tener una versión de Linux poco “pesada” con el fin de ahorrar recursos, la velocidad con la que se ejecutan ventanas y procesos en la Beagleboard es algo menor que en un PC normal. Si bien es cierto que ello no supone aparentemente ningún problema para el desarrollo de la aplicación. Todo esto resultado de la experiencia durante los meses de trabajo sobre este sistema.

### 3.2.2 Lenguaje C++

C++ es un lenguaje de programación que surgió en 1980. Fue creado por” Bjarne Stroustrup”. C++ surgió como evolución del lenguaje C con mejoras que permitieran la manipulación de objetos.

A la hora de escoger C++ como lenguaje para nuestro proyecto hay que tener las siguientes características:

- Es un lenguaje didáctico. Puede ser una base para el aprendizaje de otros lenguajes.
- Se trata de un lenguaje muy robusto.
- Permite la elaboración desde sistemas sencillos a sistemas operativos.
- Gran cantidad de documentación y tutoriales al respecto.
- Permite el manejo de punteros y memoria. Lo cual puede suponer un gran control sobre el consumo de recursos y memoria.

Por el contrario cabe mencionar que C++ aunque existen muchos entornos de programación para C++, no hay estándares para ello. Esto significa que nos podemos encontrar C++ para Windows, C++ para Linux y Mac.

Las bibliotecas usadas para el desarrollo de la aplicación, OpenCV, han sido programadas en C y C++, y teniendo los usos y aplicaciones que se les ha dado a dichas librerías, nos da una idea de la potencia de ambos lenguajes.

Para la programación en C++ sobre Beagleboard, simplemente se ha instalado un editor de texto capaz de resaltar por colores la diferente sintaxis del Lenguaje. En este caso el programa elegido fue "GEDIT".
Para la compilación, se usó el compilador de C que proporciona Linux por defecto.

### 3.2.3 Librerías

**OpenCV:**

OpenCV es una librería de distribución libre bajo licencia BSD que fue desarrollada por Intel. La primera versión apareció en 1999 y desde entonces ha sido utilizada en infinidad de aplicaciones, desde sistemas de seguridad con detección de movimiento hasta aplicaciones de control de procesos en los que es necesario el reconocimiento de objetos.

Uno de los grandes proyectos hechos con OpenCV es el vehículo no tripulado "Stanley", ganador en el año 2005 de gran Desafío DARPA.

En lo que a nosotros respecta OpenCV puede equipararse en gran medida a las extensiones de Matlab para el trabajo con imágenes.
Es por ello que se escogió el proceso mencionado de diseño en Matlab y posterior traducción a OpenCV.

Una vez terminado el diseño en Matlab y comprobado que los algoritmos planteados funcionan y entregan los resultados esperados, se pasó a traducir dicho programa.

Se puede decir que la traducción ha sido casi directa, sin embargo hay que destacar que la instalación típica de OpenCV no lleva bibliotecas que permiten el trabajo con componentes conexas como es el caso de Matlab y su "regionprops", es por ello que se escogió elegir unas librerías suplementarias desde la propia página de soporte de OpenCV (Willow garage), llamadas cvBlobsLib. Estas trabajan con elementos denominados Blobs que como hemos comentado se pueden equiparar a los mismos elementos que se calcularían en Matlab a través de "regionprops".

Esta característica de las librerías cvBlobsLib, permite un gran dinamismo a la hora de trabajar con componentes conexas y facilita mucho la labor de limpieza y aislamiento de zonas en la imagen.

A pesar de tener todo en un principio planteado y necesitar únicamente hacer una traducción entre ambos lenguajes de programación, realmente conforme se fue desarrollando la aplicación en C++ fueron apareciendo una serie de problemas y puntos a estudiar qué alargaron el desarrollo en el tiempo.

El trabajo de traducción empezó por la presentación de la imagen en tiempo real.
Se decidió hacer un solo archivo que contuviera todas las funciones en diferentes apartados, pero que se irían llamando a cada una de ellas conforme se les fuera necesitando.

Para la incorporación de OpenCV a nuestra placa, se probaron dos métodos diferentes. Aprovechando que se tenían dos tarjetas SD configuradas con Linux, en la primera se realizo descargando las librerías completas de la página de OpenCV, las cuales luego se compilaban en la carpeta deseada siguiendo las instrucciones que ellos mismos proporcionan en los archivos descargados.

Para el caso de la otra SD, aprovechando la conexión a internet y las funcionalidades del gestor de paquetes de Linux, directamente se puso en el buscador de Synaptic, "OpenCV" y se descargaron todos los paquetes asociados.

De las dos instalaciones la segunda a priori es más robusta, y tras varios meses de programación y funcionamiento no ha dado problema alguno. Sin embargo para el caso de la instalación manual, se detecto algún problema con las cámaras y la captura de JPEG.

Finalmente se decidió volcar todo el trabajo en esta segunda tarjeta SD con las instalación de OpenCV que se había realizado de manera más automatizada.

Los archivos para la compilación manual se encuentran en la carpeta de recurso, dentro del apartado "OpenCV". (archivo comprimido en .rar).

**cvBlobsLib:**

a. Librerías "cvBlobsLib".

Según la página oficial de OpenCV y la página que da soporte y documentación sobre el tema:

*"cvBlobsLib son librerías para implementar el etiquetado de componentes conexas en imágenes binarias, (similar al regionprops de Matlab). Incluyen también funciones para la manipulación filtrado y extracción de resultados de esas componentes."*

Estas trabajan con elementos denominados Blobs que como hemos comentado se pueden equiparar a los mismos elementos que se calcularían en Matlab a través de "regionprops".

Esta característica de las librerías cvBlobsLib, permite un gran dinamismo a la hora de trabajar con componentes conexas y facilita mucho la labor de limpieza y aislamiento de zonas en la imagen.

Muchos de los apartados y algoritmos empleados en el programa necesitan de las características de estas librerías para poder dar un resultado eficiente. Sin estas librerías no habría sido posible implementar la aplicación.

Como muestra del funcionamiento de estas librerías analicemos la sintaxis de estas funciones, comparándolas con las mismas en Matlab:

Para el caso de Matlab la función "regionprops" crea un vector con los valores que tienen los componentes conexos en las propiedades que nosotros hemos determinado en los argumentos de la función, en este caso, número de Euler, área, perímetro.

***propieties = regionprops(img_bw,'EulerNumber','FilledArea','FilledImage','Perimeter');***

Para trabajar con cada una de las propiedades deberemos sacar de cada uno los valores;

***euler = [propieties.EulerNumber];***
***perimetros = [propieties.Perimeter];***

En el caso de cvBlobsLib, el funcionamiento es parecido, una vez declaradas las variables, la función cBlobResult, nos devuelve un grupo de elementos dentro de una variable del tipo blob, que contiene toda la información sobre los componentes conexos calculados.

***blobs = CBlobResult( img_bw1, NULL, 255 );***

El valor que se indica al final dentro de los argumentos de la función indica el valor umbral al partir del cual se tendrán en cuenta los blobs.
Una vez se han obtenido los blobs, trabajar con ellos implica usar funciones en los que se indique la propiedad a tratar.

***blobs.Filter( blobs, B_EXCLUDE, CBlobGetArea(), B_LESS, 200 ); //Quita Blobs (Area<200)***
***blobs.Filter( blobs, B_EXCLUDE, CBlobGetMinY(), B_LESS, 2); //Quita Blobs (Borde)***
***blobs.Filter( blobs, B_EXCLUDE, CBlobGetMaxX(), B_LESS, 2); //Quita Blobs (Borde)***

Como se puede observar en las funciones superiores podemos hacer un filtro dependiendo de las propiedades de los blobs.

Las propiedades más interesantes de cara a filtrar y obtener la zona de interés son las correspondientes a las coordenadas máximas y mínimas de los “Bounding Box” de cada elemento conexo, así como área y perímetro.

Queda demostrador por tanto que en este caso cvBlobsLib viene influenciado por las prestaciones de Matlab y su Toolbox de tratamiento de imágenes.

Para la instalación de cvBlobsLib, necesitamos el código fuente que encontraremos en la carpeta de “recursos” en el apartado de cvBlobsLib.

Una vez descomprimidas en el directorio deseado necesitamos añadir la ruta a dichas librerías tal y como se especifica en el archivo Leame, que se adjunta.

### 3.2.4 Decision final; (BeagleBoard + Ubuntu + OpenCV).

Después de meses de trabajo y diferentes pruebas de rendimiento y funcionamiento, la decisión fue la de seguir aquella configuración y combinación de Hardware y Software que fuera a dar un resultado positivo.

Se deja sin desarrollo la implantación de Windows CE con OpenCV sobre Beagleboard.
Las razones fueron los conocimientos básicos que se tenía sobre Visual Studio y de WindowsCE que hacían costoso el desarrollo y correcta configuración del Sistema Operativo.
Además se quiso evitar cualquier gasto adicional en cuestión de Licencias y problemas de Copyright, por lo que el proyecto se encamino hacia el desarrollo total por Software Libre.

La modificación y uso de Linux (en nuestro caso Ubuntu), es más abierto que el caso de Windows.
Por otro lado OpenCV sobre Windows requiere una modificación exhausta de las librerías y su compilación para el caso de usarlo sobre Beagleboard.

El resultado final es un dispositivo con un sistema operativo embebido con las características reducidas para un correcto funcionamiento de las operaciones que deseamos para el correcto funcionamiento de las librerías mencionadas.

Las conclusiones obtenidas son las de un sistema estable y abierto. Manteniendo las posibilidades de mejora de rendimiento con bajo coste.

# 4 TECLADO VIRTUAL

## *4.1 Descripción.*

La aplicación central del dispositivo diseñado consiste en un teclado que funcione a través de la visión artificial. Se trata de un patrón sobre el que se situara un dedo. La cámara conectada a la Beagleboard recogerá las imágenes y será capaz de calcular la posición de ese dedo y devolver las coordenadas "reales" de dicha posición.

Se trata de una aplicación demostrativa de las capacidades de la Beagleboard así como de las librerías OpenCV, ambas de código abierto. Es por tanto un concepto o prototipo de aplicación, propensa a la evolución y desarrollo así como de la introducción de nuevas características.

Se asume por tanto que la aplicación no está completa. Además se ha recortado complejidad en ciertos aspectos, como por ejemplo el patrón. Se ha creado un patrón de 24 posiciones. Hoy en día los teclados corrientes que usamos junto a los PC contienen un número superior a 100 teclas.

A la hora de decidir la aplicación final que se desarrollaría sobre la Beagleboard se pusieron sobre la mesa distintas opciones. Se pensó en un lector de códigos BIDI, también se barajó la idea de un lector de matriculas. Finalmente se opto por hacer el teclado virtual por lo original de la idea y por cierta información recogida de internet.

Se ha detectado un incremento del uso de la visión artificial en distintos aspectos de la vida cotidiana. Desde la e-salud, hasta los mencionados lectores de matriculas en los parkings públicos, pasando por los videojuegos en primera persona que incorporan videocámaras que recogen los movimientos del jugador.
En el mundo de la telefonía móvil, se han detectado una corriente en los diseños en los que se incorporan teclados "virtuales". Se trata de teclados que son proyectados sobre las superficies a través de laser o haces de luz y situando los dedos sobre dichos teclados el dispositivo es capaz de recoger la posición y entender dicha posición como una pulsación de tecla. Este es el caso por ejemplo de los prototipos o "concept mobile" de Mozilla "Seabird" o una versión del Iphone 5 que circula por internet que ha día de hoy se desconoce si realmente lo incorporará el próximo dispositivo de telefonía de Apple.

Se trata por tanto de un software que podría tener su expansión y desarrollo en un futuro cercano.

Si bien es cierto que este dispositivo no trata de compararse ni competir con los mencionados dispositivos pero sí que podría suponer un prototipo ó concepto de cara a dispositivos embebidos que pudieran darse sobre hardware del tipo Beagleboard.

## *4.2 Características funcionales.*

La principal funcionalidad que ofrece este conjunto es el de ofrecer al usuario un periférico como un teclado habitual pero sin la necesidad de tener conectado uno.
En realidad nuestra aplicación necesita de un patrón (elemento físico) que permita la ejecución del mismo y sería equiparable a un teclado. Sin embargo este patrón podría mejorarse proyectándolo sobre una superficie como el caso de los dispositivos que hemos mencionado anteriormente. Esto sí que supone un ahorro de espacio y dinero.

La Beagleboard está pensada como herramienta didáctica, y por tanto, su uso no puede ser comparable al de un PC u ordenador portátil. En un uso profesional podríamos estar hablando de dispositivos embebidos para tareas concretas o trabajos específicos. En tales casos podría ser necesario el uso de algún teclado o mecanismo que accionamiento. Esta aplicación ofrece la posibilidad de personalizar dicho teclado en función de las necesidades. Pudiéndose ampliar o reducir la cantidad de interrupciones.

Hablamos por tanto como un complemento para dispositivos embebidos en tareas y usos concretos.

## 4.3 Implementación.

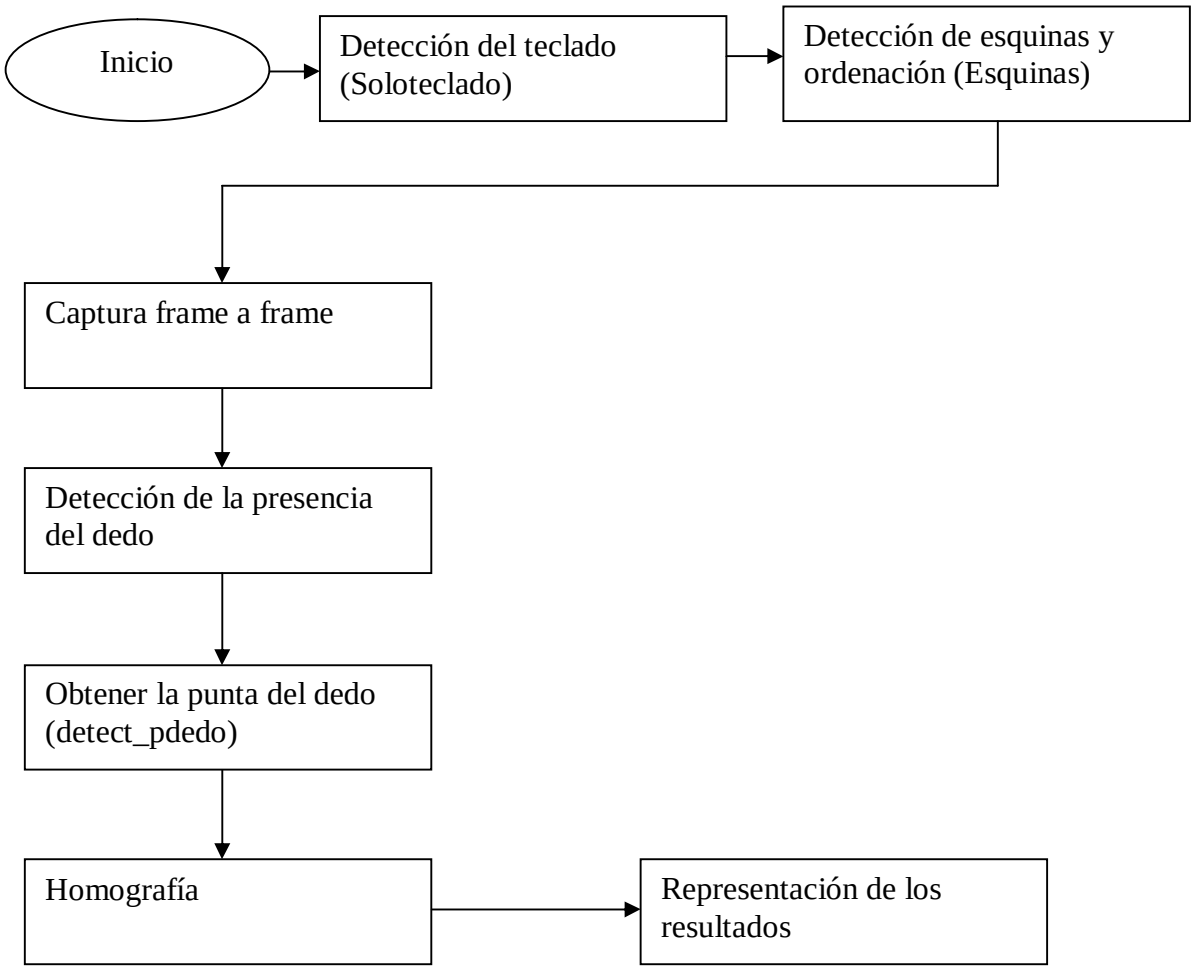
El programa está compuesto por varias subtareas, se puede decir que todas están conectadas en cascada y que la salida de una es la entrada de la siguiente.

Al iniciar el programa nos aparece una ventana en tiempo real con las imágenes que captura la cámara. Una vez tengamos el teclado en posición deberemos pulsar la barra espaciadora para que el programa empiece a funcionar de manera autónoma.
En un primer momento el programa calibra el teclado y obtiene las 4esquinas del mismo de manera ordenada.

A continuación pasa a capturar frame a frame las imágenes a la espera de que un dedo sea puesto sobre el teclado.

Así, el programa detecta dicha interrupción y pasa a calcular la posición de la punta del mismo. Finalmente con las coordenadas de la punta del dedo y las 4 esquinas calculadas en la calibración se hace un cálculo de la homografía.

El esquema de la aplicación queda de la siguiente manera:



- Soloteclado:

"Soloteclado" tiene como entradas la imagen capturada en el momento en el que se pulsa la barra espaciadora y a su salida obtendremos la zona eficaz y de interés del patrón entre todos los elementos que haya en la imagen.

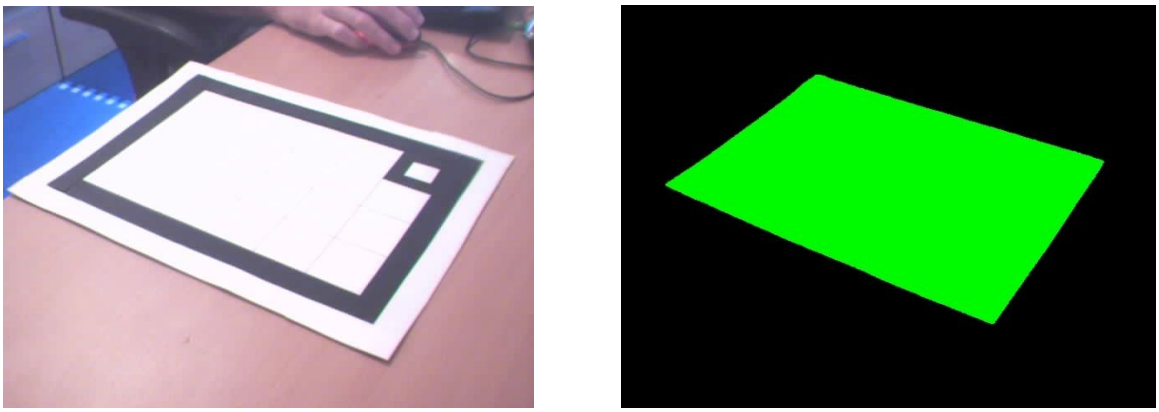
La función recoge dicha imagen, la binariza y le hace un filtrado de componentes conexas. Así nos quedamos solo con la componente conexa mencionada del teclado.

Para la binarización, teniendo en cuenta que ha sido un problema recurrente a lo largo del desarrollo del programa, usamos una función de cálculo automático del umbral, basada en el método Otsu. Esta función esta nombrada como "AutomaticThreshold". Esta función no fue implementada, se aprovechó una función ya implementada por terceros.

Para hacer el filtrado de componentes conexas nos valemos de las características que previamente le hayamos dado al patrón y que nos facilite la extracción del mismo.
Así, teniendo en cuenta la forma y color de los bordes del mismo la extracción del patrón se hace algo más sencilla.

El filtrado de los componentes conexos se hace asumiendo que la componente conexa que corresponde al teclado nunca estar tocando el borde de la imagen. Esto es, se desecha todo aquel componente que tenga un máximo "X" en 640, un mínimo "X" en 0, un máximo "Y" en 480 y un mínimo "Y" en 0.

Por otro lado y ante la posibilidad de ruido y componentes no deseadas de brillo o elementos minúsculos en la imagen, se hace un filtrado por tamaño, en el que se desechan todos aquellos Blobs o componentes conexas con un tamaño menor de 200 pixeles de área.



*Imágenes que muestran la entrada y salida a la función de "soloteclado".*

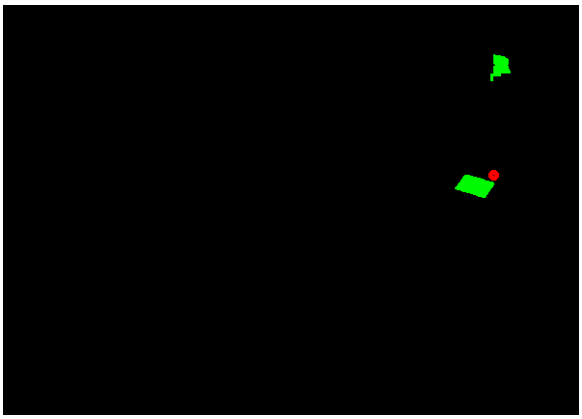
En las imágenes podemos ver la entrada a la función "Soloteclado" y la salida.

Otra de los resultados que obtenemos a través de Soloteclado, es el de conseguir el punto (0,0) de referencia.

Simplemente, aprovechando las características del patrón y la muesca que se le ha incorporado en una de los ángulos, hacemos el filtrado de Blobs, hasta que nos quedamos con ese Blob, correspondiente a un recuadro blanco de escaso tamaño.

Una vez obtenemos este, nos basta con obtener un extremo de ese recuadro, por ejemplo, haciendo un “MaxX” de ese Blob.

Posteriormente cuando obtengamos las 4 esquinas, haremos restas una a una de las esquinas con el punto de referencia, y el punto que obtenga menor resultado será el más cercano y por tanto el punto que nos interesa.



Respecto a las diferencias con la programación en Matlab, dentro de esta función cabe destacar el papel de “cvBlobsLib”. La ausencia de una función como “regionprops” en la instalación por defecto de OpenCV hace imposible la implementación de esta. Cuando se realizaron las pruebas sobre Matlab se asumía que en OpenCV contaríamos con funciones muy similares.

- Esquinas:

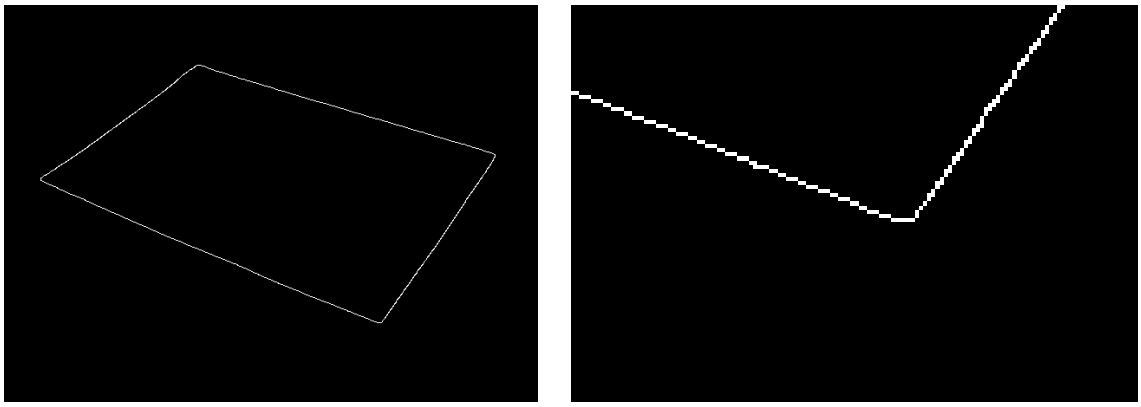
Automáticamente en cuanto la función “Soloteclado” devuelve la imagen de salida, esta es introducida en una nueva función llamada en este caso “Esquinas”.

El resultado de esquinas, devuelve la imagen en la que solo teníamos el teclado con las 4 esquinas numeradas y ordenadas en función de nuestro criterio.

El proceso para obtener este resultado es muy similar al caso de Matlab.
Se le pasa la función para obtener las líneas a través de HOUGH, posteriormente se obtienen las intersecciones de esas 4 rectas, y se filtran de manera que solo nos queden las 4 que quedan dentro del encuadre.

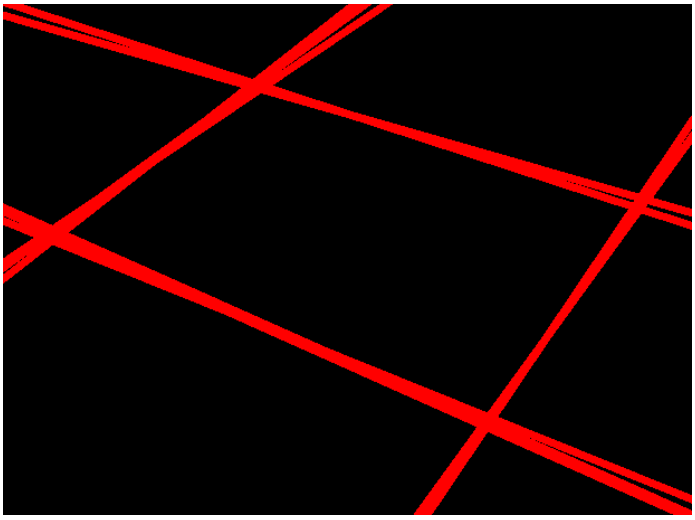
Posteriormente y con la ayuda del punto de referencia calculado en el apartado “Soloteclado.”

**Problema:** Dada la forma que adquieren los bordes del teclado al binarizarlos, las rectas que se detectan al pasar la función de Hough son más de 4.
Esto es debido a que la representación de los bordes es escalonado.



*A la izquierda el resultado de pasar "Soloteclado" por un detector de bordes Canny.*
*A la derecha el detalle del "escalonado" mencionado anteriormente.*

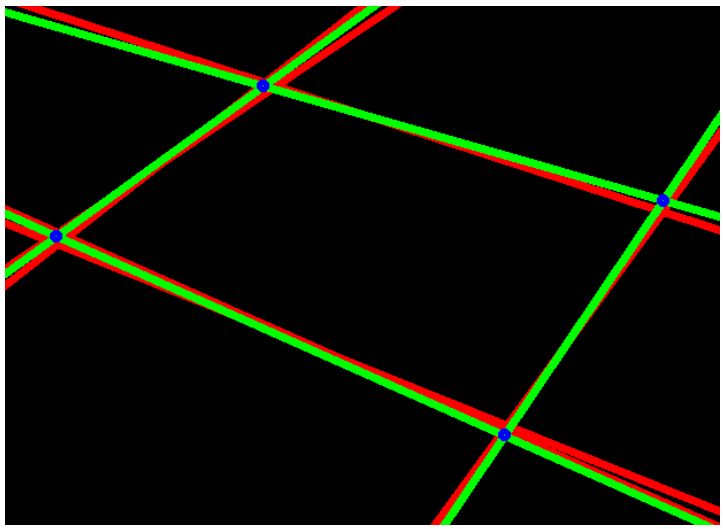
Esto lleva a que el resultado de Hough en este caso se de varias rectas de características muy parecidas:



La imagen superior muestra como se detectan más de una recta para algunos bordes del teclado.

**Solución:** Se decidió filtrar esa cantidad de rectas en función del valor de "Theta". Esto se hizo leyendo todos los valores de las distintas rectas detectadas. En el momento en que dos rectas tuvieran un calor de Theta muy similar (dependiendo de un umbral impuesto por el programador), se escogería la primera en el orden de lectura.
Esto conlleva un error de orientación según escojamos una recta u otra. Pero no afecta en absoluto el resultado final dada la escasa precisión necesaria para la lectura de la posición del dedo en comparación con el error producido.

Se barajaron también otras posibilidades como fue la de pasar el algoritmo de detección de esquinas de Harris, ya que OpenCV tiene implementada dicha función, pero los resultados no eran los suficientemente limpios como para poder aprovecharlos.



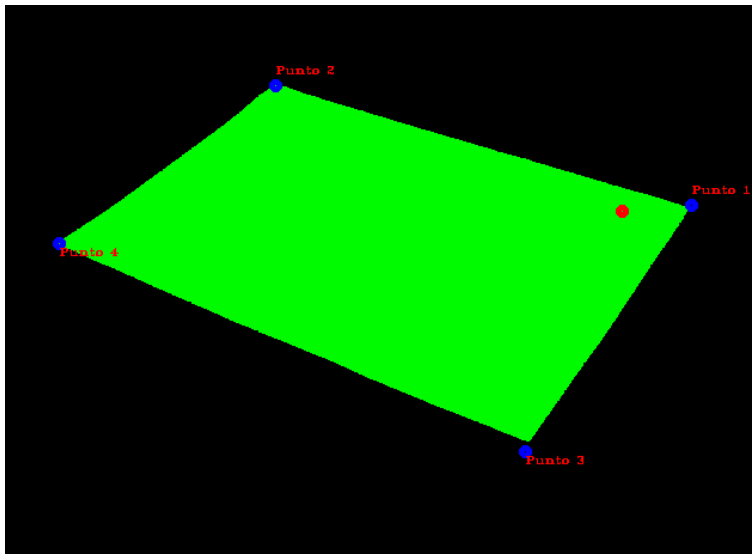
*En color verde el resultado del filtrado de las rectas con un valor de Theta muy similar.*

En este punto tenemos las rectas correctamente bien definidas. Ahora resta calcular la intersección de estas entre si, a través de la algebra.
De los puntos conocidos que pertenecen a las rectas, obtenemos la pendiente y así sacamos las ecuaciones de las rectas. Posteriormente, calculamos las intersecciones entre estas ecuaciones. Así obtenemos 6 intersecciones, de las cuales 2 quedan fuera del plano.

Y posteriormente filtrar esos resultados de manera que solo nos quedemos con aquellos que estén dentro de la cuadricula de 640 x 480.

Finalmente con la ayuda del punto de referencia calculado, podemos obtener el orden de las 4 esquinas. En un principio durante el trabajo sobre Matlab el orden de las esquinas se obtenía a través de productos vectoriales.
Durante el trabajo con OpenCV se ha detectado que el cálculo de las rectas en Hough se realiza de un determinado orden, y esto lleva a que cuando llegamos a la tarea de obtener las intersecciones estas queden siempre emparejadas las esquinas opuestas.

Esto lleva a que sabiendo únicamente cual de los 4 puntos es el que más cerca esta del punto de referencia, podamos ordenar los 4. Ya que siempre quedan emparejados el punto 1 con el 4 y el 2 con el 3.
Todas las pruebas en los que se detectan 4 esquinas correctamente, dan un orden satisfactorio.



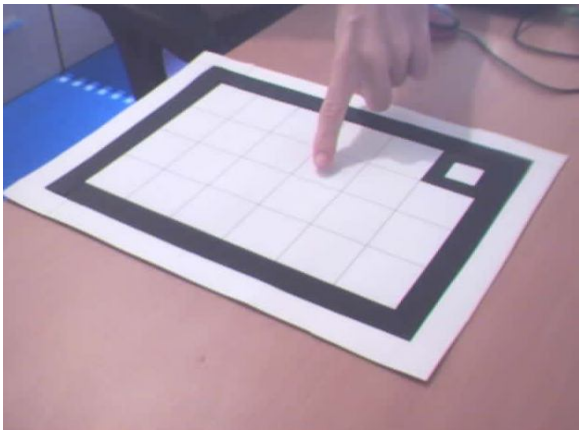
- Detect_pdedo:

Esta función es la encargada de obtener las coordenadas X e Y de la punta del dedo, las cuales marcaran el punto que estamos señalando.

La forma en la que se hace dicho cálculo es a través de la diferencia de imágenes. Se hace una comparación entre la imagen capturada al principio del programa y la que se obtiene en el momento en que el programa detecta la presencia del dedo sobre el patrón.

Se hace la resta absoluta de ambas imágenes en escala de grises, y el resultado es convertido a binario.

Nuevamente nos valemos de la herramienta de detección de Blobs para filtrar lo innecesario y quedarnos con lo que realmente nos interesa.
Para facilitar esta tarea se le aplica una máscara a la resta por la cual se hace una operación lógica de "AND" con la imagen de "soloteclado" en la que solo tenemos el bloque del teclado.



Finalmente a través de las propiedades de los Blobs obtenemos el máximo "Y" del blob del dedo. Directamente este resultado que obtenemos es el punto de interés.

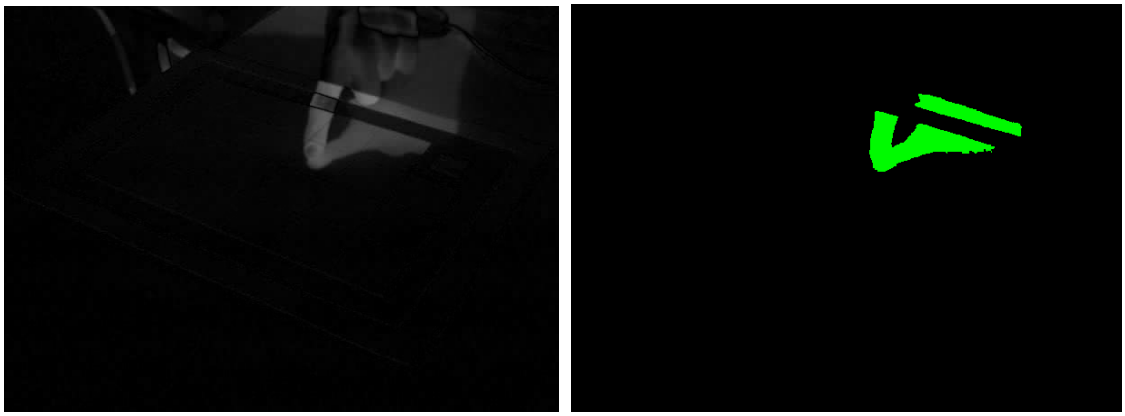
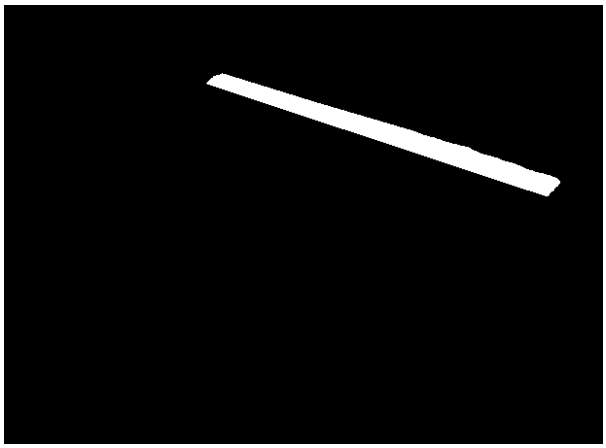


*Imagen de la izquierda refleja la resta entre imágenes en escala de grises. La de la derecha es la binarización de esta y el filtrado de lo "innecesario".*

**Problema:** Con el fin de dar autonomía al programa y que fuera automática la detección del dedo, se desarrollo un paso intermedio.

**Solución:** Este consiste en pasar cada frame captado a lo largo de la ejecución del programa por una máscara compuesta por una línea blanca de anchura fija que transcurre desde el punto 1 al punto 2.



Esta mascara se multiplica por el frame en cuestión y da como resultado el frame capturado en tiempo real reducido únicamente a esa parte del patrón.

Lo que se quiere conseguir es que al introducir el dedo en el patrón, teniendo en cuenta que se haría por ese lado, produce el "corte" de esa franja, debido a que al pasar el frame en binario, el dedo queda representado en negro mientras que la franja sigue siendo blanca.

A esta imagen obtenida se le hace un cálculo del numero de Blobs y en el caso en el que el dedo se ha introducido y ha partido dicha franja, el numero de blobs pasa de ser de 1 a 2.

Por tanto, en el momento en que un frame devuelva un numero de blobs superior a 1, el programa pasara el frame (antes de ser pasado por la máscara) a la función "Detect_pdedo".

- MHomografia:

La función de homografía es la pieza clave en el proceso del cálculo de la situación del dedo. Es la encargada de calcular la situación real del dedo respecto de las coordenadas calculadas sobre el patrón.

Se le entregan las coordenadas de las 4 esquinas sobre la imagen capturada y se compara con las coordenadas que hemos asignado en el mundo real, esto es, en nuestro caso, una cuadricula de 6x4, numerando las filas de 0 a 4 y las columnas de 0 a 6.

Para el cálculo de dicha homografía OpenCV ofrece la función **cvFindhompgraphy.**

Esta función tiene el siguiente modelo:

*cvFindHomography(ptsOrigen,ptsDestino,homografía,método,umbralRansac,mascara);*

- ptsOrigen: Son los puntos que hemos calculado como esquinas del teclado.
- ptsDestino: Son las coordenadas en el mundo, en nuestro caso: (0,0),(0,6),(4,0),(4,6).
- Homografía: Es la matriz que contendrá la homografía de ese cálculo. Sera la matriz que se multiplicara para cada punta del dedo detectada en cada caso.
- umbraRansac: Umbral que se configurara únicamente para el caso en el que hayamos escogido este método.
- Mascara: en nuestro caso nula.

Al multiplicar la matriz por el punto que marca la punta del dedo, obtenemos una coordenada comprendida para el eje x entre 0 y 6, mientras que para el eje e ira comprendido entre 0 y 4. A este valor se le hará un redondeo hacia abajo, cogiendo siempre el valor "suelo" de lo obtenido. De esta manera obtendremos las coordenadas exactas de la celda correspondiente en el patrón.

Finalmente no queda más que asignar a cada una de las 10 posibilidades (6 para el eje x y 4 para él y), unos valores que serán los que nos valgan para representar con un circulo de color sobre el patrón, la celda o "carácter" indicado.

Este proceso de representación de resultado puede ser modificado por otros como el de asignar un carácter a cada celda y representarlo en pantalla.

- Final:

Esta función es la encargada de gestionar la salida del programa. Se realizo una función aparte para así poder controlar mejor la salida a través de la pulsación de la tecla "Esc". Simplemente incorpora una serie de mensajes, anunciando la salida del programa.

## 4.4 Línea de Mejora

Hemos mencionado anteriormente que la aplicación podría desarrollarse con versionas renovadas y mejoradas. Hay ciertos aspectos de la aplicación en los que se ha detectado posibilidades de mejora.

- Patrón:
- Detección del Patrón:
- Detección del Dedo:
- Calculo de la situación de la punta del dedo.

Todos estos aspectos tienen un denominador común. Y es que uno de los grandes problemas de este dispositivo es la luz. Podemos concluir después de realizada la aplicación que uno de los puntos débiles es la luz y sus variaciones a lo largo de la ejecución del programa.

El buen funcionamiento de la aplicación depende en gran medida de la buena binarización, contraste y detección de los objetos importantes sobre el resto de elementos de la imagen.

Las razones por las que no se da correctamente esa binarización, pueden ser varias.
Las variables que entran dentro de proceso de captura y binarización de la imagen son muchos. Empezando por la cámara, la cual, debida a su reducida calidad puede producir frames con un brillo excesivo u otros en los que la exposición está por debajo de lo esperado. Además la elección del patrón no ha sido un punto demasiado desarrollo pero si se ha observado que es importante el espesor del marco de cara a la correcta captura del teclado, ya que nuevamente la reducida resolución de la cámara hace que al alejar demasiado el patrón este tienda a hacer desaparecer uno de los lados del marco, dificultando por tanto su detección.
Incluso, el material del que se compone el patrón puede afectar a la difusión de la luz y generar los molestos brillos.

El diseño del patrón podría dirigirse hacia una proyección de las teclas sobre una superficie a través de haz de luz.
Otro punto a tener en cuenta es la superficie sobre la que se apoya el patrón. Es necesario un mínimo de contraste entre ambos. Ya que si las superficies son muy similares, podría darse el caso de que se fundieran en un solo elemento.

El siguiente punto posible de modificación seria la detección del dedo. El mecanismo por el cual el programa sabe que hemos introducido un dedo es asumir que al apuntar con el dedo sobre una determinada región del teclado, estamos “rompiendo la continuidad” de la línea que transcurre desde el punto origen y el punto2. No contamos por tanto con la posibilidad de que el dedo se introduzca por la parte superior del teclado ni por los lados.

Se tiene en cuenta que al binarizar el teclado el marco queda en blanco y al introducir en el encuadre un objeto (mano), este queda binarizado en negro y por tanto se rompe la continuidad del bloque antes mencionado produciéndose en la imagen 2 componentes conexas.

El programa recoge esta característica y pasa entonces a situar exactamente dentro de las teclas, la punta del dedo.

Nuevamente nos encontramos ante el problema de que las binarización no haya sido la correcta y por las circunstancias que sean la mano quede binarizada en blanco también, haciendo imposible la rotura de la mencionada continuidad de esa parte del marco.

Por otro lado, hay que explicar que el aislamiento de esa parte del marco se hace creando una máscara con una simple franja que transcurre desde el punto origen, hasta el punto 2, con una anchura determinada. Esta anchura es fija y propuesta por el programador.
Esto lleva a que no pueda ser modificado dependiendo de la distancia cámara-marco.
Significa que estando el marco algo más alejado en el encuadre, esa parte del marco quede reflejada con una línea más estrecha y por tanto al ser multiplicada por la máscara la zona que resultará remarcada es diferente al caso en el que el marco se encuentra más cerca del objetivo y por tanto el espesor es mayor.

Una vez que el programa es consciente de la presencia de un dedo o puntero, pasa a recoger un “frame” de la situación del mismo. Lo compara con un frame de referencia que teníamos anterior en el que no existe presencia alguna de elementos que no sea el propio teclado.

Así simplemente se calcula la diferencia entre ambas imágenes en formato binario y obtenemos la parte correspondiente al dedo.
Posteriormente hacemos un barrido para obtener el punto de mayor coordenada “y” del bloque correspondiente al dedo.
Como resultado de esto se obtiene un punto de coordenadas (x,y). Esto corresponde al punto con el cual deberemos hacer el cálculo de la homografía.

La principal limitación de este sistema es que nuevamente se produzca una binarización correcta y que el bloque que corresponde al dedo sea convenientemente aislada.

# 5 PRUEBAS DE RENDIMIENTO EN BEAGLEBOARD.

El haber trabajado sobre distintas plataformas y dispositivos, permite valorar las diferencias en los resultados y las prestaciones que ofrecen cada uno de ellos.

Si bien es cierto que al eliminar Windows de la línea de trabajo no podemos llegar a valorar las diferencias que existen entre un sistema operativo y otro.

La principal diferencia que podemos encontrar, es el rendimiento que nos puede ofrecer la aplicación ejecutada sobre un PC al uso o sobre la Beagleboard.

Es de esperar que en este segundo caso al tener menor memoria y menor capacidad de procesador, los resultados que obtengamos también sean algo más lentos.
Hemos obtenido 1500 líneas de código. Una de los puntos que se tuvieron en cuenta era el de no sobrecargar la Beagleboard con trabajos de compilación. Es por eso que el código se probaba sobre un PC de sobremesa y una vez que se tenía un código estable entonces se pasaba a la placa para, en este caso, si hacer una compilación para poder tener un archivo ejecutable.

El PC de sobremesa que ha sido utilizado a lo largo del desarrollo de la aplicación es un ordenador HP con Pentium 4 de procesador y 256 MB de Ram.

La Beagleboard tiene la misma Memoria RAM pero sin embargo posee menor capacidad de procesador.

La acción de compilar las 1500 líneas de código en el caso del PC tarda alrededor de los 10 segundos mientras que la Beableboard tarda casi 15.
Por otro lado durante la ejecución del programa se ha detectado que la captura de frames en un caso se hace más lento que en el otro. Al PC hacer un ciclo de 10 frames le cuesta 7 segundos, mientras que la Beagleboard tarda 13 segundos.

Hay que destacar que el programa incorpora algunas pausas entre frame y frame para no cargar demasiado la aplicación y poder ver mejor el proceso. Es posible reducir estos tiempos, siempre teniendo en cuenta el rendimiento final.

Existen diferencias de rendimiento entre uno y otro dispositivo pero las prestaciones que tenemos con la Beagleboard en el reducido espacio que conlleva compensan esas diferencias que en el resultado final no afectan tanto.